Softwarearchitektur der gitterbasierten Sensordatenfusion des Projekts Stadtpilot

Sebastian Ohl
Richard Matthaei
Matthias Müller
Markus Maurer

Braunschweig : Institut für Regelungstechnik, 2011

Softwarearchitektur der gitterbasierten Sensordatenfusion des Projekts Stadtpilot

Sebastian Ohl Richard Matthaei
Matthias Müller Markus Maurer

Freitag 31 Dezember, 2010

1 Einleitung

Aufbauend auf den Erfahrungen aus dem Projekt CarOLO der TU Braunschweig (Rauskolb u. a., 2008) und der damit verbundenen erfolgreichen Teilnahme an den Wettbewerben der DARPA Urban Challenge werden die Aktivitäten des langfristig angelegten Forschungsvorhabens Stadtpilot vom abgeschlossenen Testgelände mit definiertem Verkehr in den realen innerstädtischen Verkehr verlagert. Im Sommer 2010 hat der Versuchsträger bereits die ersten Testfahrten mit autonomer Längs- und Querführung auf dem Braunschweiger Stadtring erfolgreich absolviert.

Das reale urbane Umfeld bietet dabei neue Herausforderungen, wobei für die Umfeldwahrnehmung im Wesentlichen die erhöhte Eigengeschwindigkeit (von ca 30 km/h auf bis zu 60 km/h bei Überholmanövern) und die damit verbundene kürzere Sichtbarkeit von Hindernissen sowie die deutlich erhöhte Anzahl an Fremdfahrzeugen zu nennen sind. Vergleichbar zu den Bedingungen bei der DARPA Urban Challenge ist die Komplexität der statischen und unstrukturierten Randbebauung.

Um das Fahrzeugumfeld maschinell wahrnehmen zu können, ist der eingesetzte Versuchsträger mit diversen Sensoren zur Umfeldwahrnehmung ausgestattet. Diese werden u. a. mithilfe einer gitterbasierten Sensordatenfusion zu einem einheitlichen Bild der Umgebung fusioniert (Thrun u. a., 2005) und über eine Schnittstelle zur Umfeldwahrnehmung einer Applikation zur Verfügung gestellt.

Da der Versuchsträger nicht nur im Projekt Stadtpilot, sondern auch in verwandten Projekten des Instituts eingesetzt (Saust u. a., 2010) und beständig weiterentwickelt wird, muss die Softwarearchitektur der Umfeldwahrnehmung flexibel einsetzbar ausgelegt werden.

In verschiedenen Beiträgen wurden bereits Architekturen für Sensordatenfusion vorgestellt (Scheunert u. a., 2008; Waltz und Llinas, 1990; Carvalho und Heinzelman, 2003; Baig u. a., 2009). Alle teilen die Verarbeitung in unterschiedliche mehr oder weniger detaillierte Ebenen ein und betrachten dabei meistens die gesamte Verarbeitungskette eines autonomen Systems. Eine verbreitete allgemeine Architektur zur Sensordatenfusion ist das funktionale Modell der Joint Director of Laboratories (JDL) des US Department of Defense (Steinberg u. a., 1999), das insgesamt fünf Ebenen vorsieht, die über einen gemeinsamen Datenbus kommunizieren. Durch die Betrachtung der Architektur auf einem hohen Level eignen sie sich jedoch nur bedingt zur direkten Implementierung einer Sensordatenfusion.

In diesem Beitrag wird die Architektur zur gitterbasierten Sensordatenfusion des Projekts Stadtpilot vorgestellt. Diese Architektur stellt ein einfach zu nutzendes objektorientiertes Framework zur Erstellung von gitterbasierten Sensordatenfusionen basierend auf unterschiedlichen Sensoren bereit. Durch die Festlegung von definierten Schnittstellen zwischen den einzelnen Elementen der Architektur bis auf Ebene einzelner Klassen kann eine einfache Austauschbarkeit von Algorithmen und Verarbeitungsstufen sichergestellt werden.

Zunächst wird in Abschnitt 2 kurz auf den Kontext der Architektur eingegangen. Darauf folgend beschreibt Abschnitt 3 allgemein die Architektur der gitterbasierten Sensordatenfusion sowie die Aufteilung in unterschiedliche Layer. Anschließend werden die Schnittstellen zur Decodierung der Kommunikationsprotokolle der Sensoren beschrieben. Abschnitt 5 geht schließlich auf die Schnittstellen und Komponenten eines Grid-Moduls innerhalb der Fusionsebene ein. Abschließend wird in Abschnitt 6 die derzeitige Ausprägung der Architektur beschrieben.

2 Architekturkontext

Die in diesem Beitrag beschriebene Architektur spezifiziert eine Sensordatenfusion zur Umfeldwahrnehmung in einem Straßenfahrzeug, die zum Ziel hat, mehrere Messwerte von einem oder mehreren Sensoren durch eine Gitterstruktur abzubilden. Dabei ist es unerheblich, welche Vorarbeitungsschritte die Messwerte vorher bereits durchlaufen haben. Eine Gitterstruktur wird dabei beschrieben als die Abbildung einer realen Fläche durch die Zusammenfassung von zweidimensionalen Bereichen (Zellen) deren Ausmaße durch eine mathematische Vorschrift festgelegt werden (z. B. äquidistantes Gitter, radiales Gitter). Diese Gitterstruktur besitzt ein eigenes Koordinatensystem, welches Abstände in der Einheit [Zelle] misst. Ferner ist diese Gitterstruktur relativ zu einem festen Punkt eines anderen Koordinatensystems, z. B. Weltkoordinaten- oder Fahrzeugkoordinaten, definiert.

Da diese Architektur sich explizit an gitterbasierte Sensordatenfusionen richtet, werden beispielsweise Objekthypothesen basierte Sensordatenfusionen (Ohl und Maurer, 2011) und Highlevel-Fusions-Systeme, die Entscheidungen unterschiedlicher Agenten zusammenführen (Rosenblatt, 1997) oder Situationen bewerten (Freyer u. a., 2007), nicht durch die Architektur abgedeckt.

3 Softwarearchitektur der Fahrzeugumfeldwahrnehmung

Unter Softwarearchitektur wird im Allgemeinen Folgendes verstanden:

> "Die Softwarearchitektur eines Systems beschreibt dessen Software-Struktur respektive dessen -Stukturen, dessen Software-Bausteine sowie deren sichtbare Eigenschaften und Beziehungen zueinander." (Vogel u. a., 2009, Seite 48)

Die Festlegung einer Softwarearchitektur beginnt mit der Definition von Anforderungen. Aufgrund der unterschiedlichen Rahmenbedingungen, durch den Einsatz in verschiedenen Projekten, müssen unterschiedliche Sensoren und Schnittstellen genutzt bzw. bedient werden können. Bedingt durch den Einsatz in der Forschung sollen einzelne Algorithmen innerhalb der Verarbeitung einfach tauschbar sein, um den Einsatz von neuen Verfahren zu vereinfachen.

Zusammenfassend ergeben sich daraus drei Anforderungen (Reihenfolge entsprechend des Textes):

1. Unterschiedliche Sensorkonfigurationen mit unterschiedlichen Software- und Hardware-Schnittstellen (z. B. CAN, LIN, Flexray, Ethernet) müssen genutzt werden können.
2. Die Unabhängigkeit der einzelnen Verarbeitungsstufen und Algorithmen innerhalb der Sensordatenfusion soll zum Zwecke der Austauschbarkeit gewährleistet sein.
3. Der Einsatz in unterschiedlichen Projekten soll mit geringem Aufwand möglich sein.

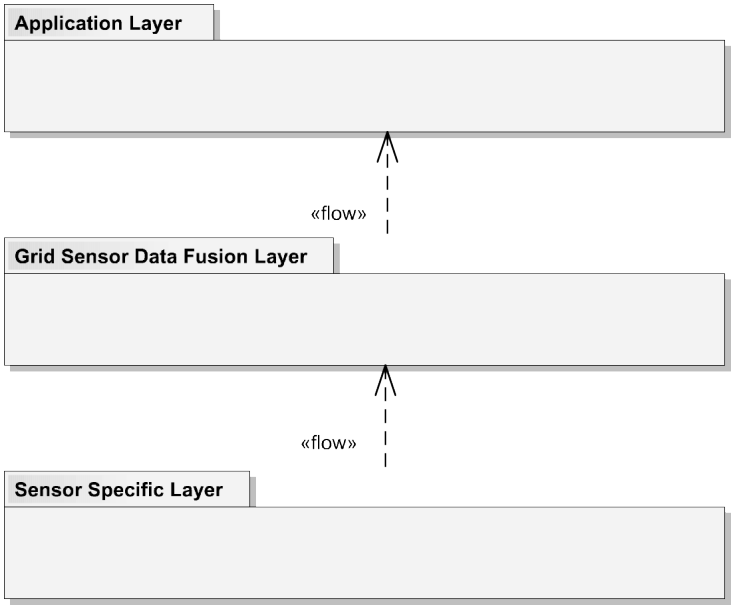

Abbildung 1: Paketdiagramm der Architektur

Bezug nehmend auf die JDL-Architektur aus Waltz und Llinas (1990) wird diese Architektur in mehrere Ebenen aufgeteilt. Abbildung 1 bildet die Ebenen durch UML-Pakete ab. Die Sensorebene (*Sensor Specific Layer*) entspricht dem Level 0 der JDL-Architektur und enthält die Verarbeitung der sensorspezifischen Protokolle und Bussysteme sowie eine ggf. notwendige Vorverarbeitung (z. B. Koordinatentransformationen). Die zweite Ebene, entsprechend Level 1, wird als Fusionsebene (*Grid Sensor Data Fusion Layer*) bezeichnet. Ihre Aufgabe ist es, die von der Sensorebene erzeugten Daten zu verarbeiten und der Ebene der Applikation zur Verfügung zu stellen. Diese letzte Ebene fasst die Level 2-4 der JDL-Architektur zusammen und enthält die eigentliche Funktion des Fahrzeugs (*Application Layer*) wie beispielsweise eine automatische Notbremsfunktion zur Reduzierung von Unfallfolgen (Reichel u. a., 2010).

Um die Schnittstellen so allgemein wie möglich und dennoch austauschbar zu gestalten, wird ein Container-Format, *UnifiedPointCloud*, für Messpunktwolken definiert. Dieses enthält, neben den Daten über die Position des Sensorkoordinatensystems, eine Menge von Messpunkten in einem kartesischen 3D-Koordinatensystem.

Je sensorspezifischem Protokoll wird innerhalb der Sensorebene eine eigene Komponente (z. B. *IBEOAlaskaSensorDeEncoder*) erzeugt. Sie besitzt einen sensorspezifischen Eingang zum Empfang von Messwerten und sensorspezifischen Ausgang zur Stimulation des Sensors (z. B. mit Daten zur Fahrzeugeigenbewegung), sowie einen Ausgang an dem das Containerformat bereitsteht. Durch diese Vereinheitlichung der Schnittstellen wird Anforderung 1 erfüllt, da sich nun Sensoren austauschen lassen, ohne Schnittstellen anpassen zu müssen.

Die Fusionsebene enthält, je nach Anforderungen der Applikation, unterschiedliche Sensordatenfusionen. Jede stellt ein Modul da, das nach dem gleichen Schema aufgebaut ist (siehe Abschnitt 5) und Messdaten mithilfe einer oder mehrerer Gitterstrukturen verarbeitet. Die Zusammenfassung aller enthaltenen Gitterstrukturen wird im Folgenden als Grid bezeichnet, eine einzelne Gitterstruktur wird als Layer bezeichnet. Nach dem in Thrun u. a. (2005) beschriebenen Algorithmus lässt sich eine gitterbasierte Sensordatenfusion in die Komponenten (inverses) Sensormodell, Filter sowie Datenspeicherung aufteilen. Diese Basiselemente werden um eine Komponente zur Präsentation der Daten (*View*) gegenüber der Applikation erweitert. Die einzelnen Schnittstellen zwischen diesen Komponenten werden in Abschnitt 5 erläutert. Sie ermöglichen die Umsetzung der 2. Anforderung bezüglich der weitgehenden Unabhängigkeit zwischen den Algorithmen.

Die Applikationsebene verarbeitet die in der Fusionsebene oder Sensorebene erzeugten Daten und stellt auf ihrer Basis die gewünschte Funktion dar. Durch die zwei der Applikationsebene unterlagerten Ebenen sowie die Einführung der Präsentations-Komponente wird die Applikation soweit von der eigentlichen Sensorik entkoppelt, dass die Bearbeitung von unterschiedlichen Projekten in einem Versuchsträger (Anforderung 3) erheblich vereinfacht wird.

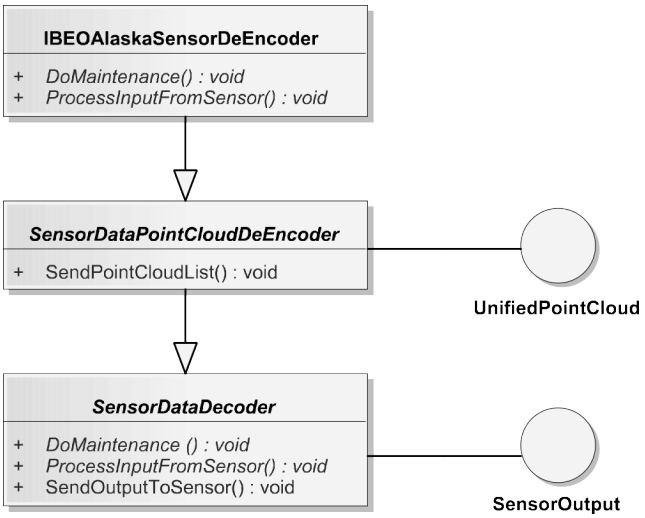

Abbildung 2: Klassendiagramm der sensorspezifischen Ebene

4 Sensorebene

Jeder Sensortyp wird von einer speziell auf den Sensor angepassten Klasse (z. B. *IBEOAlaskaSensorDeEncoder*) verarbeitet. Diese wird von der abstrakten Basisklasse *SensorDataPointCloudDeEncoder* abgeleitet. Sie stellt Servicefunktionen (z. B. Koordinatentransformationen) sowie einen groben Rahmen zur Verarbeitung von Sensordaten bereit. Der Empfang sowie der Versand von Datenpaketen wird bereits hier bzw. durch die Basisklasse *SensorDataDecoder* erledigt. Abbildung 2 zeigt ein Klassendiagramm der Komponente am Beispiel eines IBEO Alaska XT Sensors. Die konkrete Sensor-Komponente widmet sich der De- bzw. Encodierung des sensorspezifischen Protokolls sowie den sensorspezifischen Aufgaben (z. B. Bereitstellen von Fahrzeugeigendaten). Der bereits erwähnte Sensordatencontainer *UnifiedPointCloud* bildet die Schnittstelle zur nächsten Ebene. In ihm werden die gemessenen Punkte des Sensors abgelegt.

5 Fusionsebene

Die Fusionsebene enthält, je nach Anforderungen der Applikation, mehrere Fusionsmodule. Je nach verarbeitender Anwendung auf der Applikationsebene können aufwendige Algorithmen eingesetzt oder nur das Umformen von unterschiedlichen Eingangsdaten zu einem einheitlichen Format durchgeführt werden. Im Folgenden wird kurz auf die grundlegenden Abläufe innerhalb eines Fusionsmoduls eingegangen. Anschließend werden die

internen Strukturen der enthaltenen Komponenten näher erläutert.

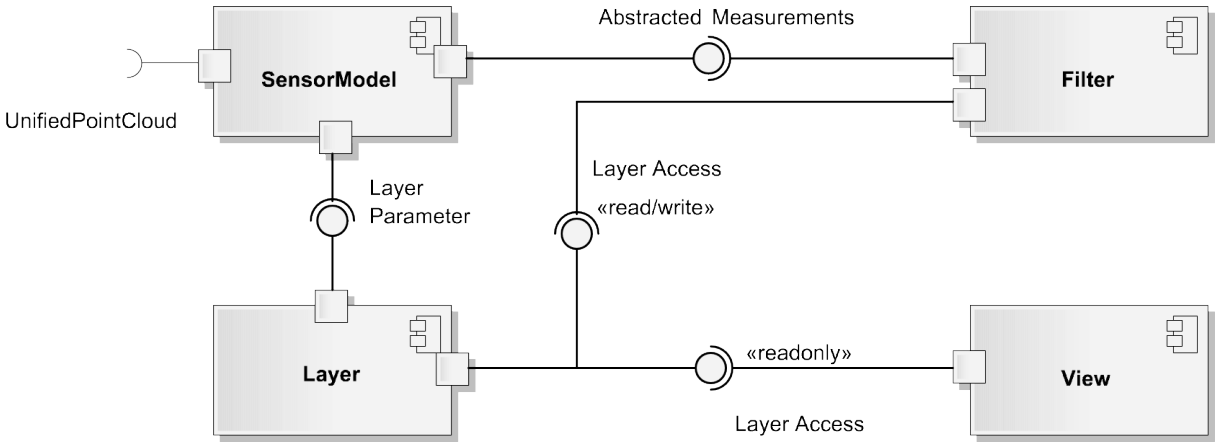

Abbildung 3: Komponentendiagramm der Fusionsebene

Die einzelnen Bestandteile dieser Ebene wurden bereits in Abschnitt 3 vorgestellt. In Abbildung 3 sind sie in ihrem Zusammenspiel dargestellt. Ein Fusionsmodul ist nach dem Pipes-And-Filter-Muster (Buschmann u. a., 1996) aufgebaut. Jede der Verarbeitungsstufen arbeitet parallel zu den anderen und kommuniziert die Ergebnisse seiner Algorithmen über definierte Schnittstellen. Dieses Prinzip ermöglicht eine Wiederverwendung der einzelnen Algorithmen in unterschiedlichen Grid-Modulen. So kann beispielsweise ein fertiges Sensormodell in ein Fusionsmodul mit einem experimentellen Filtertyp eingebracht werden.

Innerhalb eines Grid-Moduls können alle Komponenten mehrfach vorkommen. Dies ermöglicht den Einsatz von Layern mit z. B. unterschiedlichen Auflösungen sowie die Nutzung unterschiedlicher Eigenschaften von Sensoren (z. B. Höhenauflösung und Hindernisexistenz). Zwischen zwei Fusionsmodulen wird unterschieden, wenn sie eine unterschiedliche Zielsetzung haben. Wird beispielsweise der gleiche Bereich um das Fahrzeug mit dem gleichen Zweck (z. B. Hinderniswahrnehmung) abgebildet, so wird von einem Modul gesprochen. Hat das eine Modul jedoch die Aufgabe der Hinderniswahrnehmung und das andere zum Ziel eine Höhenkarte der Umgebung zu bestimmen, so wird von unterschiedlichen Modulen gesprochen.

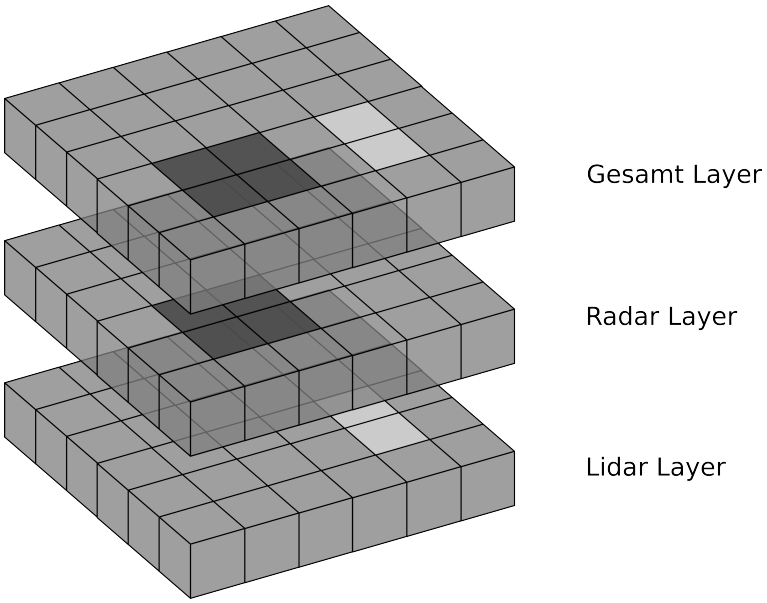

Abbildung 4: Logische Ebenen (Layer), z. B. Radar Layer, Lidar Layer und Gesamt Layer

5.1 Layer-Komponente

Die Layer-Komponente bildet die zentrale Komponente innerhalb der gitterbasierten Fusion. Sie speichert und verwaltet die eigentlichen Daten. Eine Komponente stellt dabei eine einzelne Gitterstruktur dar. Diese kann jedoch in Beziehung zu anderen Gitterstrukturen stehen. Projiziert man die Beziehungen in die dritte Dimension, so kann von unterschiedlichen logischen Ebenen (Layern) gesprochen werden. Im realen Einsatz kann mit diesen Layer-Komponenten beispielsweise eine dezentrale Sensordatenfusion (Darms, 2007) aufgebaut werden. Hierbei könnte jeder Sensor zunächst von einer Layer-Komponente mit der für ihn optimalen Auflösung verarbeitet werden, anschließend würden diese Daten, die u. U. den gleichen Bereich um das Fahrzeug abdecken, miteinander kombiniert werden. Abbildung 4 zeigt ein solches Beispiel. Hierbei werden ein Radar- und ein Lidarsensor zunächst von einer Layer-Komponente verarbeitet, um anschließend diese Daten miteinander zu kombinieren.

In Abbildung 5 ist das Klassendiagramm der Layer-Komponente dargestellt. Eine *Layer*-Klasse besitzt mehrere Eigenschaften, die es ihm ermöglichen, die Verwaltung der Datenstruktur zu übernehmen. Wie in Abschnitt 2 beschrieben, besitzt jeder *Layer* einen Ursprung bezüglichen eines Koordinatensystems, zusätzlich werden Ausdehnung und Anzahl der Gitterzellen gespeichert. Um ein möglichst paralleles Arbeiten sowie ein einfaches Verschieben und Erweitern des abgedeckten Bereichs zu ermöglichen, wurde die Daten-

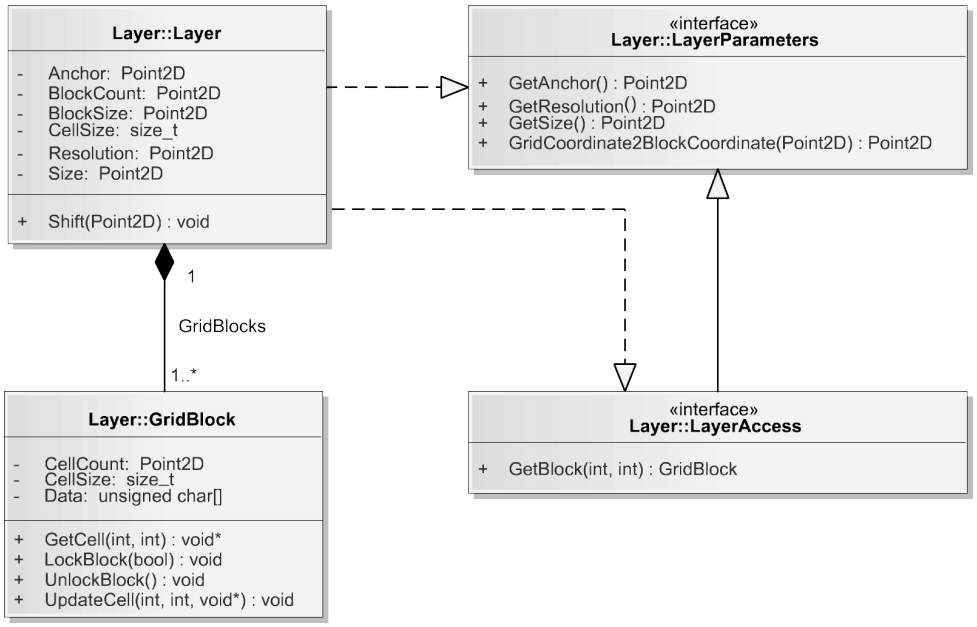

Abbildung 5: Klassendiagramm der Komponente Layer

struktur zusätzlich in Grid-Blöcke, unterteilt. Diese stellen Mikro-Grids dar und übernehmen die eigentliche Speicherung der Daten (siehe Abbildung 6). Jeder Grid-Block lässt sich einzeln für den Schreib- und Lesezugriff sperren. Hierdurch können beispielsweise zwei Filterkomponenten gleichzeitig Daten eintragen, wenn ihre Sichtbereiche sich nicht überschneiden. Die beiden angebotenen Schnittstellen des *Layers* werden von den anderen Komponenten genutzt, um beispielsweise Koordinaten zu transformieren oder Zellen zu aktualisieren.

5.2 Sensormodell-Komponente

Ein Sensormodell stellt die Abbildung von Messwerten auf die zur Aktualisierung einer Zelle benötigten Größen dar (Scheunert u. a., 2008). Je nach Sensortyp werden spezielle Modelle benötigt. Insgesamt lassen sich vor allem Unterscheidungen nach der eingesetzten Technologie treffen. So muss aufgrund eines Lasermesswerts eine andere Menge von Zellen mit anderen Werten aktualisiert werden, als es beispielsweise für die Verarbeitung eines Radarmesswerts nötig wäre.

Die abstrakte Klasse *SensorModel* mit den zugehörigen Schnittstellen ist in Abbildung 7 dargestellt. Die Schnittstelle zur Filter-Komponente bildet eine Liste von zu aktualisierenden Zellen zusammen mit einem Filter spezifischen Datentyp. Stellvertretend

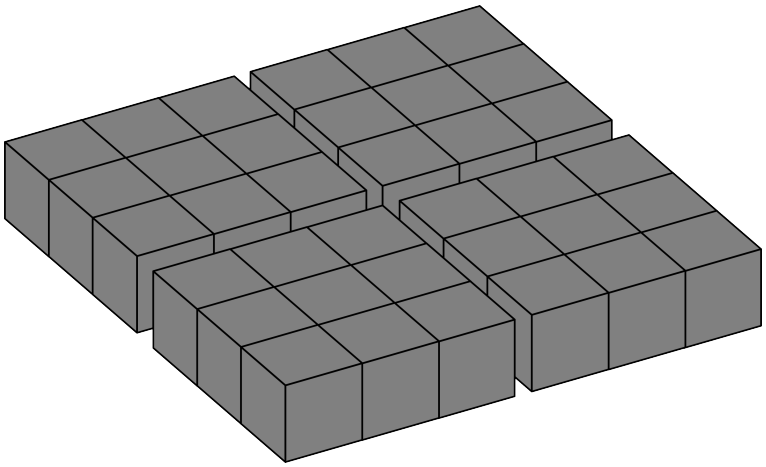

Abbildung 6: Speicherabbild eines aus mehreren Grid-Blöcke bestehenden Layers

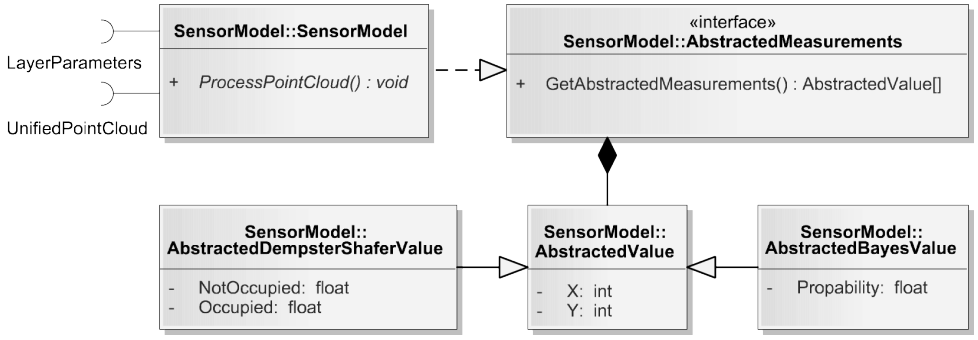

Abbildung 7: Klassendiagramm der Komponente Sensormodell

sind hier Eingangsgrößen für ein Dempster-Shafer-Filter und ein Bayes-Filter abgebildet. An einen Filter übergeben wird immer die Gesamtliste der aufgrund einer *UnifiedPointCloud* erzeugten Zellaktualisierungen. So ist sichergestellt, dass die Daten in einer Layer-Komponente konsistent sind.

Da die Datenstruktur *UnifiedPointCloud* lediglich Messpunkte enthält, aber keine Informationen über das zugehörige Sensorkoordinatensystem umfasst, muss diese Information zur korrekten Zellaktualisierung über die Sensoreigenschaften des Sensormodells abgebildet werden. Am Beispiel eines rotierenden Laserscanners mit einem radialen Koordinatensystem, der in einer Layer-Komponente mit Bayes-Filter eingetragen werden soll, würde das Verfahren folgendermaßen ablaufen: Innerhalb der *UnifiedPointCloud* sind alle durch den Sensor gemessenen Laserreflexe enthalten. Messwerte, die jedoch außerhalb der maximalen Reichweite des Sensors liegen, sind hier nicht gespeichert. Soll nun ein Layer auf Basis eines Messpunkts der *UnifiedPointCloud* aktualisiert werden, so wird der

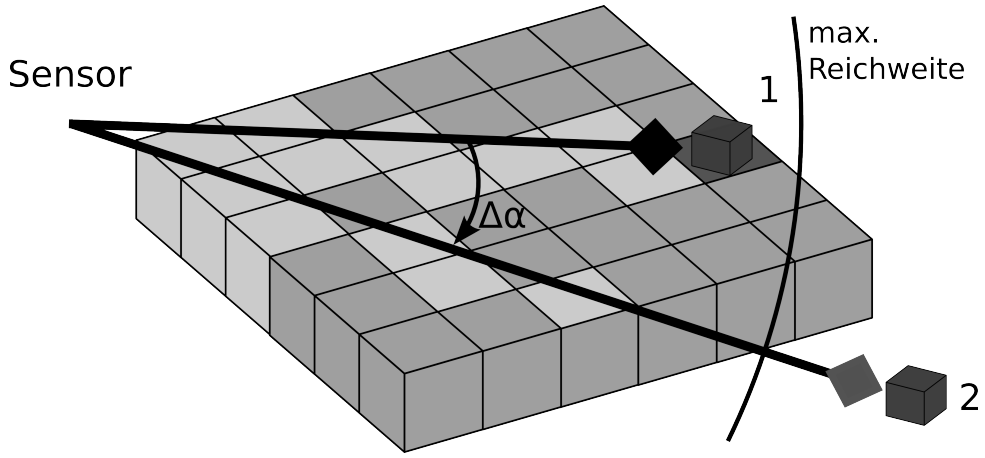

Abbildung 8: Beispiel der Verarbeitung eines rotierenden Laserscanners in einem Sensormodell mit Belegt- (dunkelgrau), Frei-Eigenschaft (hellgrau), Objekt 1 innerhalb, Objekt 2 außerhalb der Sensorreichweite

Strahlengang vom Ursprung der Messung bis zu dem Messpunkt als frei und der Messpunkt selbst aber als ,,belegt“ angenommen (siehe Abbildung 8, Objekt 1). Strahlen ohne Laserreflex oder mit einem Laserreflex außerhalb der gewünschten maximalen Reichweite müssen durch das Sensormodell bestimmt werden. Dies lässt sich über eine Analyse der Winkelauflösung ($\Delta\alpha$) von vorhandenen Laserreflexen realisieren. Die so gefundenen Strahlen, die zwar real existieren aber keinen Laserreflex geliefert haben, werden bis zur gewünschten maximalen Reichweite als ,,frei“ aktualisiert (siehe Abbildung 8, Objekt 2).

5.3 Filter-Komponente

Die Filter-Komponente trägt die Ergebnisse der Sensormodelle in den Zellen einer konkreten Layer-Komponente ein. Hierzu kann beispielsweise das Verfahren von Dempster-Shafer oder von Bayes verwendet werden. Das Filter bestimmt die eigentliche Datenstruktur einer Layer-Komponente, da das Filter die einzige Komponente ist, die Daten in einer Layer-Komponente ablegen darf. Bei der Verarbeitung der Daten eines Sensormodells ist auf eine geeignete Vorsortierung zu achten, um ein möglichst paralleles Arbeiten zu ermöglichen. Da jeder Grid-Block einzeln mit einem Mutex gesichert ist, lassen sich durch eine auf Grid-Blöcke bezogene Sortierung mehrere Datensätze gleichzeitig eintragen.

Abbildung 9 bildet die Klassenstruktur der Filter-Komponente ab. Zu sehen sind ferner die genutzten Schnittstellen zum Zugriff auf eine Layer-Komponente sowie zum Empfang

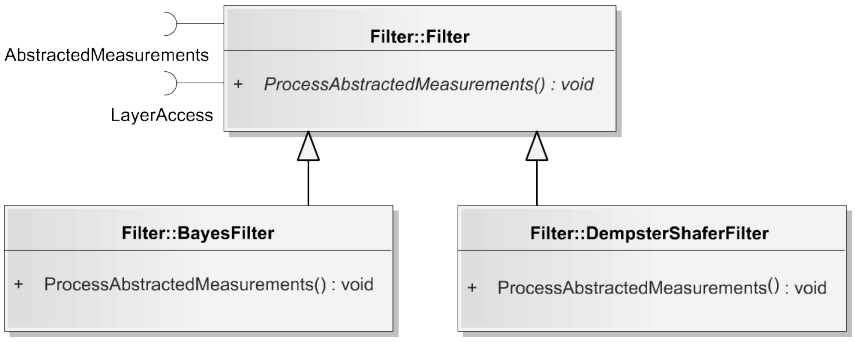

Abbildung 9: Klassendiagramm der Komponente Filter

der Ergebnisse eines Sensormodells.

5.4 Präsentations-Komponente

In einer Layer-Komponente aggregierte Daten müssen zur späteren Weiterverarbeitung in einer Applikation oft noch konvertiert oder vorverarbeitet werden, um beispielsweise befahrbare Bereiche zu detektieren (Leonard u. a., 2008). Die Gitterstruktur dazu zunächst an die Applikation zu übermitteln, ist in der Regel nicht empfehlenswert, da hier sehr große Datenmengen übertragen werden müssen. Die Präsentations-Komponente (View-Komponente) stellt deshalb eine spezielle Sichtweise auf eine Layer-Komponente da. In dieser können aufwendige Umformungen direkt auf den Datenstrukturen der Komponente durchgeführt und speziell an die Applikation angepasst werden.

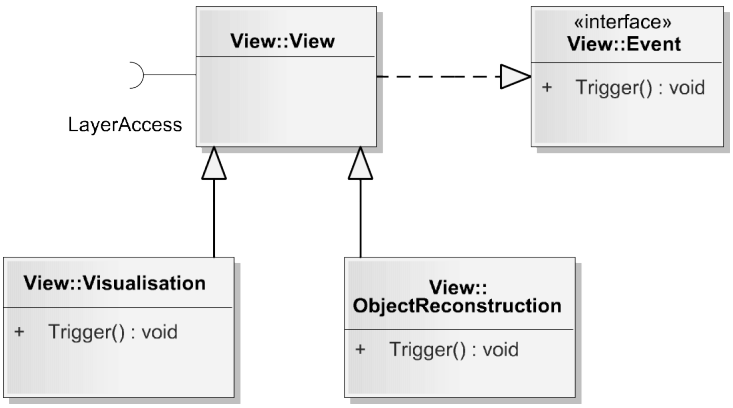

Abbildung 10: Klassendiagramm der Komponente View

Das Klassendiagramm der Präsentations-Komponente ist in Abbildung 10 dargestellt. Die Basisklasse *View* wird dabei durch ein externes Ereignis (z. B. zeitgesteuert) aktiviert

und führt anschließend die Umformung aus. Denkbar sind hier beispielsweise Verfahren zur Objektrekonstruktion, die Bestimmung des befahrbaren Bereichs oder auch das Kopieren der gesamten Daten einer Layer-Komponente oder lediglich veränderter Grid-Blöcke zur Darstellung in einer Visualisierung.

6 Einsatz im Projekt Stadtpilot

Die in den vorangegangenen Abschnitten vorgestellten Komponenten wurden im Rahmen des Projekts Stadtpilot in der Programmiersprache C++ umgesetzt. Implementiert wurde ein Sensormodell für Laserscanner, das unterschiedlich parametrisiert werden kann. Das Ergebnis dieses Modells lässt sich mit einem Bayes Filter fusionieren. Die eingesetzte Datenstruktur ist in der Lage sich mit dem Fahrzeug mitzubewegen oder auch ortsfest eingesetzt zu werden.

Der Versuchsträger des Projekts Stadtpilot ist mit Laser- und Radarsensoren ausgerüstet worden (siehe Abbildung 11). Für die gitterbasierte Fusion werden bislang nur die Lasersensoren genutzt. Im vorderen Bereich des Fahrzeugs sind zwei Laserscanner vom Typ IBEO Alaska XT angebracht. Das Heck des Fahrzeugs ist mit einem Laserscanner vom Typ IBEO Lux ausgerüstet. An zentraler Position auf dem Fahrzeugdach ist ferner ein Laserscanner vom Typ Velodyne HDL-64ES2 montiert.

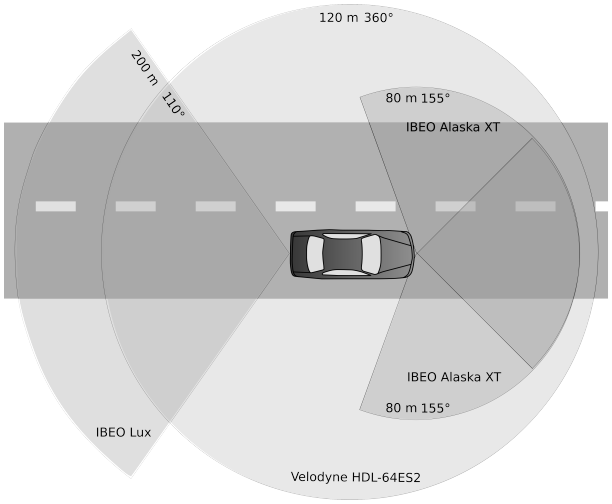

Abbildung 11: Gitterbasierte Sensorkonfiguration Versuchsträgers des Projekt Stadtpilot “Leonie”

Die Befestigungspositionen und Blickwinkel richten sich nach dem Einsatzgebiet sowie

den auftretenden Fahrsituationen. Ein besonderes Augenmerk liegt auf dem Front- sowie Heckbereich des Fahrzeugs. Hier ist eine mehrfache Redundanz der Abdeckungsbereiche durch unterschiedliche Sensorhersteller gegeben. Die Messdaten der Sensoren werden miteinander fusioniert, um die Rate der Fehldetektionen möglichst gering zu halten.

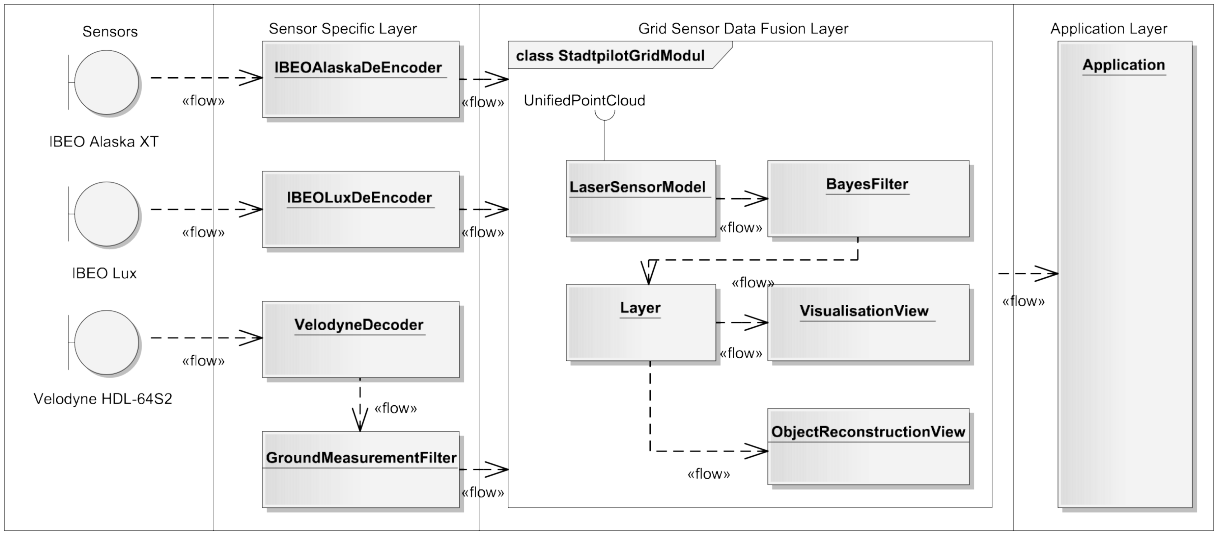

Abbildung 12: Ausprägung der Architektur im Projekt Stadtpilot

Abbildung 12 stellt die genutzten Sensoren, die implementierten Module und die Komponenten in ihrem Zusammenhang dar. Die eingesetzte Layer-Komponente spannt eine Fläche von 100m x 100m bei einer Auflösung von 20mm x 20mm um das Fahrzeug auf. Aufgeteilt ist der Layer insgesamt 8 x 8 Grid-Blöcke *à* 64 x 64 Zellen. Derzeitig werden von den Komponenten drei Sensoren verarbeitet, die in dieser gemeinsamen Layer-Komponente fusioniert werden. Das Sensormodell zur Verarbeitung von rotierenden Lasersensoren lässt sich in unterschiedlichen Parametrierungen hierzu einsetzen. Zur Verarbeitung der Messwerte des Velodyne HDL-64ES2 wird ferner eine Vorverarbeitung benötigt, um Messwerte des Bodens zu entfernen. Zur Präsentation des Grids gegenüber der Applikation wird aktuell das gesamte Grid kopiert. Diese Schnittstelle wird auch zur Visualisierung eingesetzt. Zusätzlich wird momentan eine Komponente zur Objektrekonstruktion entwickelt.

7 Zusammenfassung

Durch die Verfügbarkeit von bestehenden Algorithmen und die Definition der Schnittstellen innerhalb eines Grid-Moduls (siehe Abschnitt 5) lassen sich einfach Erweiterungen in

bestehende und neu zu erstellende Fusionssysteme einbringen (Anforderung 2). Der Einsatz von unterschiedlichen Präsentationsalgorithmen ermöglicht die Nutzung eines Umfeldwahrnehmungssystems mit den Applikationen unterschiedlicher Projekte (Anforderung 3). Durch Vereinheitlichung der Schnittstellen zur Sensorprotokolldecodierung (siehe Abschnitt 4) wurde auch die Anforderung 1 nach dem Einsatz unterschiedlicher Sensorkonfigurationen umgesetzt.

Insgesamt wurde die Umsetzung der Sensordatenfusionen des Projekts Stadtpilot gegenüber dem Projekt CarOLO als deutlich einfacher wahrgenommen. Der Einsatz von weiterentwickelten Algorithmen ließ sich leichter umsetzen, während durch die strikte Definition der Schnittstellen zwischen und innerhalb der drei Ebenen die Wiederverwendung von Algorithmen und Komponenten in unterschiedlichen Projekten mit unterschiedlichen Sensorkonfigurationen möglich wurde.

Literatur

[Baig u. a. 2009] Baig, Q. ; Vu, T.-D. ; Aycard, O.: Online localization and mapping with moving objects detection in dynamic outdoor environments. In: *Intelligent Computer Communication and Processing, 2009. ICCP 2009. IEEE 5th International Conference on*, 2009, S. 401 –408

[Buschmann u. a. 1996] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-oriented software architecture - A system of patterns*. Wiley, 1996

[Carvalho und Heinzelman 2003] Carvalho, Hervaldo S. ; Heinzelman, Wendi B.: A general data fusion architecture. In: *In Int. Conf. on Info. Fusion*, 2003, S. 1465–1472

[Darms 2007] Darms, Michael: *Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren für Fahrerassistenzsysteme*. Düsseldorf, TU Darmstadt, Dissertation, Januar 2007

[Freyer u. a. 2007] Freyer, J. ; Deml, B. ; Maurer, M. ; Färber, B.: ACC with enhanced situation awareness to reduce behavior adaptions in lane change situations. In: *Intelligent Vehicles Symposium, Istanbul* IEEE (Veranst.), 2007. – ISSN 1-4244-1067-3

[Leonard u. a. 2008] Leonard, John ; How, Jonathan ; Teller, Seth ; Berger, Mitch ; Campbell, Stefan ; Fiore, Gaston ; Fletcher, Luke ; Frazzoli, Emilio ; Huang, Albert ; Karaman, Sertac ; Koch, Olivier ; Kuwata, Yoshiaki ; Moore, David ; Olson, Edwin ; Peters, Steve ; Teo, Justin ; Truax, Robert ; Walter, Matthew ; Barrett, David ; Epstein, Alexander ; Maheloni, Keoni ; Moyer, Katy ; Jones, Troy ; Buckley, Ryan ; Antone, Matthew ; Galejs, Robert ; Krishnamurthy, Siddhartha ; Williams, Jonathan: A perception-driven autonomous urban vehicle. In: *Journal of Field Robotics* 25 (2008), August, Nr. 9, S. 727–774

[Ohl und Maurer 2011] Ohl, Sebstian ; Maurer, Markus: Ein Kontur schätzendes Kalmanfilter mithilfe der Evidenztheorie. In: *7. Workshop Fahrassistenzsysteme - FAS 2011*, 2011. – angenommen

[Rauskolb u. a. 2008] Rauskolb, Fred W. ; Berger, Kai ; Lipski, Christian ; Magnor, Marcus ; Cornelsen, Karsten ; Effertz, Jan ; Form, Thomas ; Graefe, Fabian ; Ohl, Sebastian ; Schumacher, Walter ; Wille, Jörn-Marten ; Hecker, Peter ; Nothdurft, Tobias ; Doering, Michael ; Homeier, Kai ; Morgenroth, Johannes ; Wolf, Lars ; Basarke, Christian ; Berger, Christian ; Gülke, Tim ; Klose, Felix ; Rumpe, Bernhard: Caroline: An autonomously driving vehicle for urban environments. In: *Journal of Field Robotics* 25 (2008), August, Nr. 9, S. 674–724

[Reichel u. a. 2010] Reichel, M. ; Bouzouraa, Mohamed E. ; Siegel, Andreas ; Siedersberger, Karl-Heinz ; Maurer, Markus: Erweiterte Umfelderkennung und Nutzung einer Ausweichanalyse als Grundlage einer aktiven Gefahrenbremsung. In: *AAET 2010, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel* Gesamtzentrum für Verkehr Braunschweig e.V. (Veranst.), 2010, S. 150–169

[Rosenblatt 1997] Rosenblatt, Julio: *DAMN: A Distributed Architecture for Mobile Navigation*. Pittsburgh, PA, Robotics Institute, Carnegie Mellon University, Dissertation, Januar 1997

[Saust u. a. 2010] Saust, Falko ; Bley, Oliver ; Kutzner, Ralf ; Wille, Jörn M. ; Friedrich, Bernhard ; Maurer, Markus: Exploitability of vehicle related sensor data in cooperative systems. In: *International Conference on Intelligent Transportation Systems*, 2010

[Scheunert u. a. 2008] Scheunert, U. ; Mattern, N. ; Lindner, P. ; Wanielik, G.: Generalized Grid Framework for multi sensor data fusion. In: *Information Fusion, 2008 11th International Conference on*, 2008, S. 1 –7

[Steinberg u. a. 1999] Steinberg, Alan N. ; Bowman, Christopher L. ; White, Franklin E.: Revisions to the JDL data fusion model, SPIE, 1999, S. 430–441

[Thrun u. a. 2005] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic robotics*. MIT Press, September 2005 (Intelligent robotics and autonomous agents)

[Vogel u. a. 2009] Vogel, O. ; Arnold, I. ; Chugthai, A. ; Ihler, E. ; Kehrer, T. ; Mehlig, U. ; Zdun, U.: *Software-Architektur - Grundlagen - Konzepte - Praxis*. 2. Spektrum Akademischer Verlag, 2009

[Waltz und Llinas 1990] Waltz, Edward L. ; Llinas, James: *Multisensor Data Fusion*. Norwood, MA, USA : Artech House, Inc., 1990